10 tips with PowerShell in Windows Server 2008 - Part 1

In fact, there are a lot of Windows Server 2008 tasks that we can do a lot faster with PowerShell than the GUI-based application or tool. In the following article, we will introduce you some basic and most frequently used operations with PowerShell ...

TipsMake.com - In fact, there are quite a few tasks of Windows Server 2008 that we can do a lot faster with PowerShell than the GUI application or support tool. In the following article, we will introduce you some basic and most frequently used operations with **PowerShell** :

- Change the administrator account password - **Administrator**

- Restart or turn off the server

- Restart 1 any service

- Turn off any process

- Create a report

1. Change admin password with PowerShell:

Assume that you are logged in with an **Administrator** domain account on a **Windows 7 Desktop** computer on the domain system. The requirement here is to change the password on the **remote server** in Chicago named **CHI-WIN7-22** . After an account is used many times, the chances of 'unlocking' the password are higher. That is why we should change the password of the account after a period of use.

First, we need to create a new ADSI object for the **local Administrator** account on the server. To do this, type the following command at the **PowerShell** screen:

[ADSI] $ Admin = 'WinNT: // CHI-WIN7-22 / Administrator'

In essence, the above command will perform 'retrieve' admin account on machine **CHI-WIN7-22** and assign it to **ADSI** object named **$ Admin** . If you want to connect to another computer, you just need to change **CHI-WIN7-22** to the corresponding name of that computer. If you want to know how long that password has been used to change it, we will use the command:

$ Admin.PasswordAge

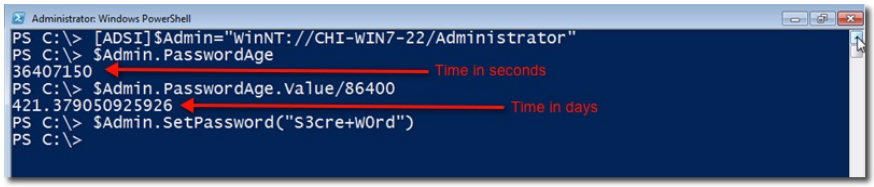
The above statement will display the amount of time that has elapsed since the last change, and since the return value is calculated in seconds, we will divide by **86,400** to convert to date:

$ Admin.PasswordAge.Value / 86400

If you pay close attention, we will see that we use the **Value** attribute here. That's because the **PasswordAge** information is stored in the **collection** form, so we need to know the value of that collection to check the exact number to perform the division calculation. Finally, change the password using the **SetPassword** function, along with the new password - **argument** . For example, if you want to set a new password, **S3cre + WOrd** , you must type the following command:

$ Admin.SetPassword ('S3cre + WOrd')

However, you should note that after pressing Enter, the system will not display any messages, but this change will take effect immediately. Because we used the method - method, not the cmdlet - command. Unlike **cmdlets** , **SetPassword** does not support **-whatif** or **-confirm** .



2. Turn off or restart the server:

To do this, we will use a WMI-based **cmdlet** pair of commands, which is **Restart-Computer** and **Stop-Computer** . Although ink doesn't go too far into analyzing each of these commands, we still have to mention, since those cmdlet commands accept alternative information - great for specifying user accounts already. login is available, through which we can execute the corresponding commands of that account.

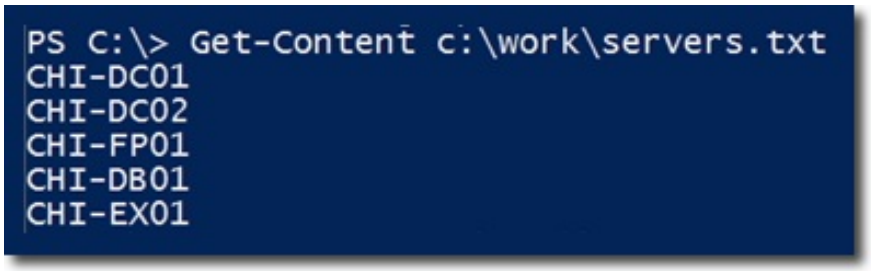
Besides, you can take advantage of **whatif** - **confirm** and - **confirm** , which means that if you want to execute the command to shut down or restart the system, we can do it in our own way. And if applied in practice, especially in the case of having to turn off multiple computers at the same time, this will be the most effective solution. The basic structure of this command is:

Restart-Computer -ComputerName

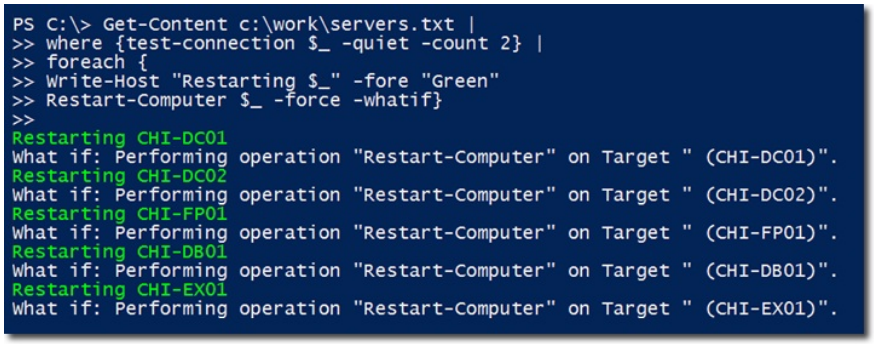
with - **ComputerName** is an array of string data, assigned with the name of 1 or more computers. **Stop-Computer** uses the same syntax, for example if you want to restart 2 computers named **CHI-DC02** and **CHI-FP01** , we type:

Restart-Computer 'CHI-DC02', 'CHI-FP01'

The -whatif parameter is used to show what happens when the command is executed
Now, we will move on to a more complicated example, assuming we have a list of multiple computers in a file called **servers.txt** , using the **Get-Content** cmdlet command to get the content inside. in that text file:



So if you have to do it on multiple computers, enter the names of those computers into a text file. And each time we need to reboot, we just need to apply the **Get-Content** cmdlet command. Examples are as follows:



Next, the system will push the list part to the where clause to check. Inside this where clause, we will proceed with the **test-connection** command - the main purpose is to ping each individual computer. The - **quiet** parameter will return **true** or **false** , while - **count 2** means that each computer can only ping 2 times. For each machine that is pinged twice, the system will automatically browse them.

Next, we will have to use **foreach** . Specifically, for each computer name that passes the above check, the system will display a message with a green color that says: ' **Restarting** '. The parameter **$ _** represents each object in the system, and then the **Restart-Computer** cmdlet command will be invoked to restart all computers that have been pinged. Besides, we can also use the - **force** parameter to prevent any account from logging in.

And finally, continue to use **whatif** - to show the entire activity, progress or happenings happening in the system.

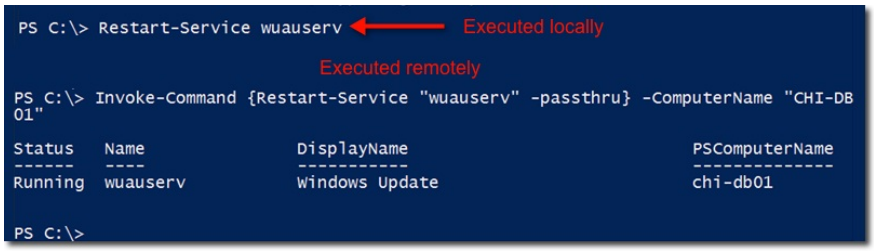
---

3. Restart the service:

Here, **Restart-Service** will be the cmdlet command used to restart the service - service in the system. Although the cmdlet command does not have a mechanism to support the connection to a remote computer, **PowerShell Remoting** can be activated for use via the remote mechanism.

To do this, just type the command: **Restart-Service 'service',** where ' **service** ' is the name of the service that needs to be restarted. On the other hand, if you want to apply this on one or more remote computers, use the cmdlet **Invoke-Command** and **PowerShell Remoting** commands **.**

For example, in the screenshot of **PowerShell** below, there are 2 cases where we have performed Restart-Service syntax to restart the service called **wuauserv** (of **Windows Update**). First, the **Restart-Service** is executed directly on the system, but then this command continues to be executed on a database server called **CHI-DB01** with the help of **Invoke-Command.**

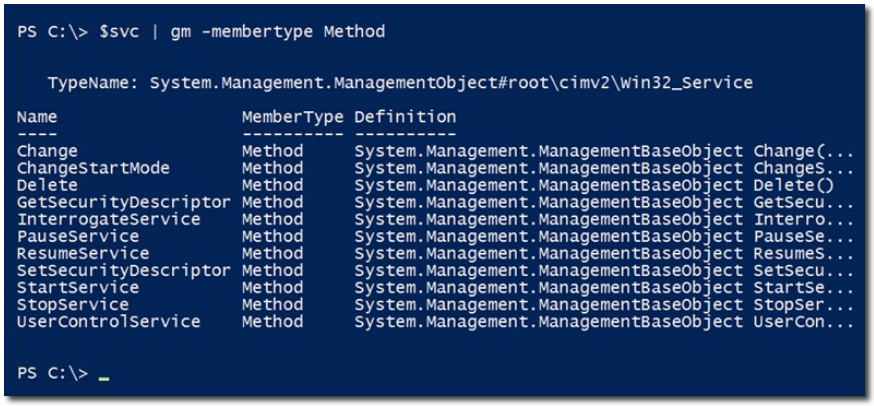


By default, **Restart-Service** will not save any objects into the system unless you use the command - **passthru** . Therefore, the information we see below (**Status** , **Name** .) is the result after using - **passthru** . If the service works on many different computers, and wants to restart that service, we just need to add the names of those computers to the list, separated by commas.

Another way to do this is to use **WMI** . First, we will create a new **WMI** object:

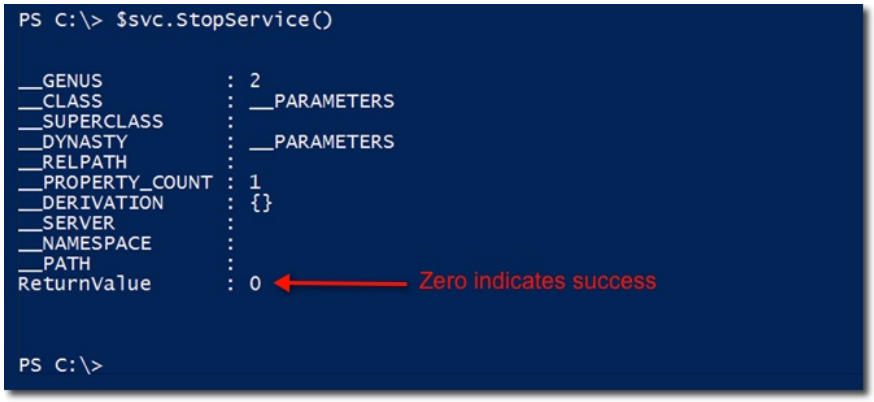
In which **gwmi** is the alias of **Get-WmiObject.**

Specifically, the system will first push objects to **Get-Member** (with **alias gm**):

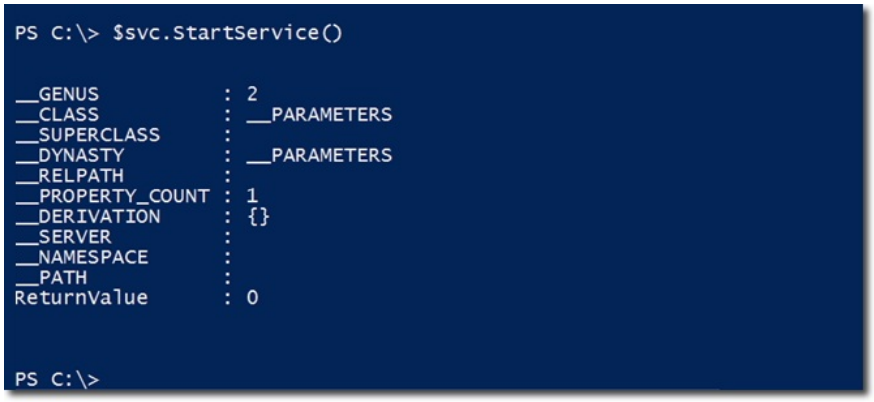


If you pay close attention, you will find that there is no way to restart the service. That means we will have to use the **StopService** command to stop, then start with **StartService** .

Here's how to stop the service using the **object** 's **StopService** . If you get a **ReturnValue** value of 0, it means that service has been stopped. In case you get another value, find that value on MSDN:



To start the service, we use the **StartService** command:



To test, you type the corresponding **get-service** command on the computer. This command allows the user to connect to the remote computer:

4. Turn off any process:

This is also the most often done on multiple servers, and to do this we will have to use the **Stop-Process** cmdlet command. Similarly, if you want to apply on the remote system, you will need to go to **Stop-Process** that comes with **PowerShell Remoting.**

In fact, we have two ways to do this with the **Stop-Process** cmdlet **.**

The first way is quite simple, you just need to run the **Stop-Process** command, and then the name or corresponding ID of the process. For example, with the screenshot below, the process name to shut down here is

Calc (which is the **Windows Calculator**):

```
PS C:\> Calc
PS C:\> get-process calc

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     79      17     6148      10816    80     0.11   2556 calc


PS C:\> stop-process -name calc
PS C:\>
```

Next, we will switch to the application case on the **remote** computer. The example here is the process of **notepad** on the **remote** machine called **chi-fp01.**

```
PS C:\> invoke-wmimethod -path win32_process -name create -argumentlist notepad.exe
 -computername chi-fp01


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ProcessId        : 1760
ReturnValue      : 0
```

Then, we have to check if the application is working by using the **ps** - alias parameter of **Get-Process:**

```
PS C:\> ps notepad -Comp chi-fp01

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     52       6     1276       3700    42               1760 notepad
```

And when you have the remote process to process, we will apply some of the same steps as in the above reboot, you use the **Invoke-Command** and **PowerShell Remoting** commands to perform **Stop** syntax. **-Process** on remote **chi-fp01** server **.**

```
PS C:\> invoke-command {ps notepad | kill} -comp chi-fp01
PS C:\> _
```

---

5. Create report Disk Utilization:

System administrators must regularly monitor the status of the server as well as the available space on the server's drive. In fact, we can easily do this by using **WMI** or **Win32_LogicalDisk** class **.**

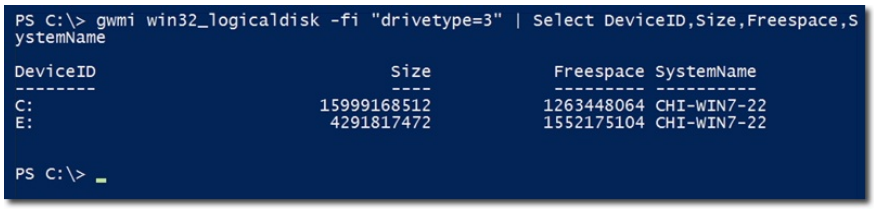
With **WMI** , it is possible to execute query commands directly on local or remote machines, 1 or more computers. Besides being able to save data into CVS file or directly into database, HTML page or simplest is displayed on the screen.

Example of a simple WMI statement on a local machine:

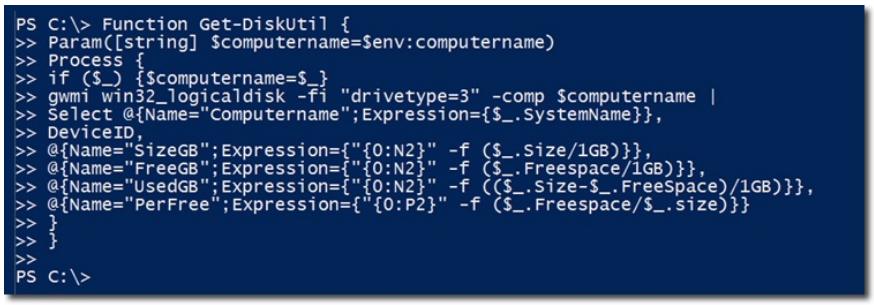
Get-WmiObject win32_logicaldisk -filter 'drivetype = 3' | Out-File c: ReportsDisks.txt

Specifically, we use the cmdlet **GetWmiObject** command to display information returned from the **Win32_LogicalDisk** class. Then there is the - **filter** to filter **drivetype** related **information = 3** - corresponding to the fixed system partitions like drive C. This also means that other devices such as USB Flash drives, map drives The system will not be listed here, and this information will be saved in the **Disks.txt** file **.**

In the screenshot below, we specify that the displayed results will include some information such as **Device ID, Size, Space .:**



Specifically, we will create a new function function here called **Get-DiskUtil.** Depending on the actual situation in which you can put this function in the script file, then download to the profile, or load from many other scripts to use in combination.



The above function will use the computer name as a parameter, and the default value is the name of the local computer.

Here, we continue to use the **Process** script to prevent the name of the local computer from entering the function.

Next is **GetWmiObject** :

```
>> gwmi win32_logicaldisk -fi "drivetype=3" -comp $computername |
```

The result of this process is focusing on the **Select-Object** cmdlet command (with the corresponding alias **Select**). Next, we take advantage of **Hashtable** to create a properties object called **Computername** - with the main function of renaming **SystemName** of the current object **($ _)** to **Computername** , while **DeviceID** will be preserved:

```
>> Select @{Name="Computername";Expression={$_.SystemName}},
>> DeviceID,
>>
```

After that, we continue to create 1 **Hashtable** object pair, first with the **Size** property, broken down into 1GB separate parts, so change it to **SizeGB** . Next is **Freespace** with the same basic formula:

```
>> @{Name="SizeGB";Expression={"{0:N2}" -f ($_.Size/1GB)}},
>> @{Name="FreeGB";Expression={"{0:N2}" -f ($_.Freespace/1GB)}},
>>
```

To continue, we will create a new property named **UsedGB** - not available in WMI, this is simply the difference between **Size** and **FreeSpace** , then split by 1GB:

```
>> @{Name="UsedGB";Expression={"{0:N2}" -f (($_.Size-$_.FreeSpace)/1GB)}},
>>
```

And finally, we need to create another attribute called **PerFree** - understand what is 'percent free', with the main function is to display the free space of the drive in%:

```
>> @{Name="PerFree";Expression={"{0:P2}" -f ($_.Freespace/$_.size)}}
>>
```

Below is a screenshot of the system when we enter the name of the computer, display it in a tabular format - **Format-Table** or ft, then the results will automatically be adjusted to size by using -auto.

```
>>
PS C:\> Get-DiskUtil chi-fp01 | ft -auto

Computername DeviceID SizeGB FreeGB UsedGB PerFree
------------ -------- ------ ------ ------ -------
CHI-FP01     C:       19.90  7.67   12.23  38.53 %
CHI-FP01     F:       100.00 99.91  0.09   99.91 %


PS C:\>
```

In the next step, we will move on to the details of creating a report on **Disk Utilization** status for the server. The first thing to do here is to save all data into the $ data variable, so that the user will not have to type the command manually anymore. After that, the system continues to push the results obtained to the where object, execute the ping command to check, assign the computer name to the newly created function - **Get-DiskUtil.**

And when the system displays the interface as shown below, it means the data storage has been completed:

```
PS C:\> $data=gc c:\work\servers.txt | where {Test-Connection $_ -quiet -count 2} |
 Get-DiskUtil
PS C:\>
```

Data information has been successfully stored in **$ data** . If desired, the user can define this information section in **$ data** and sort by **computername** variable, besides it can be sent via **Out-Printer** or **Out-File:**

```
PS C:\> $data | sort computername | ft -auto

Computername DeviceID SizeGB FreeGB UsedGB PerFree
------------ -------- ------ ------ ------ -------
CHI-DB01     C:       39.90  18.46  21.44  46.27 %
CHI-DB01     D:       10.00  9.92   0.08   99.23 %
CHI-DB01     E:       10.00  9.88   0.12   98.80 %
CHI-DC01     C:       14.90  8.62   6.28   57.83 %
CHI-DC02     C:       11.90  4.92   6.98   41.38 %
CHI-EX01     C:       39.90  22.55  17.35  56.52 %
CHI-FP01     F:       100.00 99.91  0.09   99.91 %
CHI-FP01     C:       19.90  7.67   12.23  38.53 %


PS C:\>
```

Similarly, if you want to download that information to **SQL database** or **Excel** , we only need to convert the format - convert to **CVS** file as follows:

```
PS C:\> $data | export-csv c:\work\diskutil.csv
PS C:\>
```

If you import the **CSV** file, you can store the temporary state of the partitions and the drive when the command is executed:

```
PS C:\> Import-csv c:\work\diskutil.csv
```

For example:

```
UsedGB       : 0.09
PerFree      : 99.91 %

Computername : CHI-DB01
DeviceID     : C:
SizeGB       : 39.90
FreeGB       : 18.46
UsedGB       : 21.44
PerFree      : 46.27 %

Computername : CHI-DB01
DeviceID     : D:
SizeGB       : 10.00
FreeGB       : 9.92
UsedGB       : 0.08
PerFree      : 99.23 %

Computername : CHI-DB01
DeviceID     : E:
SizeGB       : 10.00
FreeGB       : 9.88
UsedGB       : 0.12
PerFree      : 98.80 %

Computername : CHI-EX01
DeviceID     : C:
SizeGB       : 39.90
FreeGB       : 22.55
UsedGB       : 17.35
PerFree      : 56.52 %
```

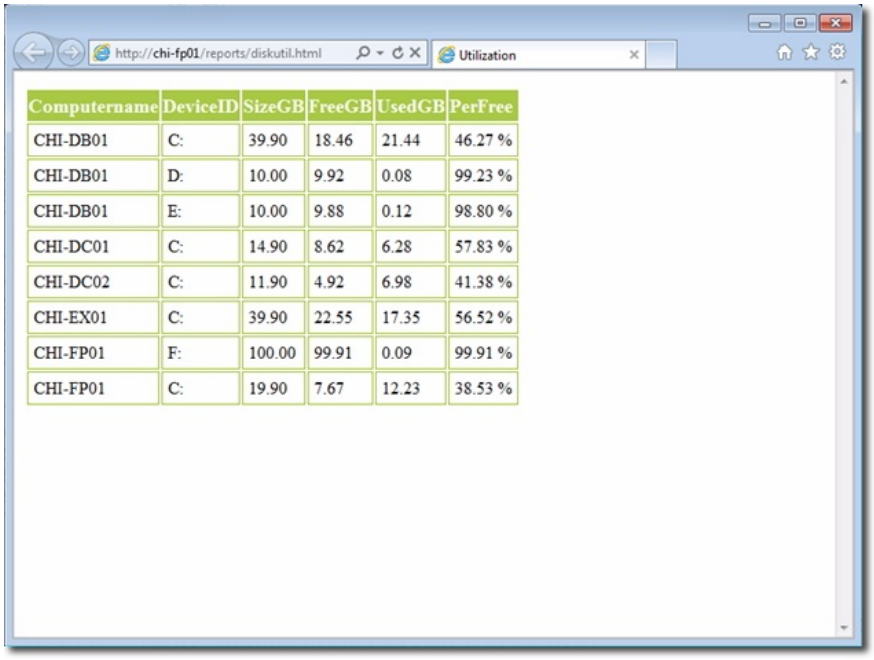
And finally, we will discuss how to create a report in HTML format. Similar to the above, you start by filtering **$ data** and assigning **Sort Computername** , then passing this information to **ConvertTo-HTML** cmdlet command, besides naming and the path to the CSS file. This CSS section is really necessary because **ConverToHTML** does not perform any formatting operations nor sort information. Then, create the resulting file according to the structure as below:

```
PS C:\> $data | Sort Computername |
>> ConvertTo-HTML -Title "Utilization" -Css "http://chi-fp01/styles/sample.css" |
>> Out-File "\\chi-fp01\webreports$\diskutil.html"
>> _
```

Then, type **start** to start:

```
PS C:\> start "http:\\chi-fp01\reports\diskutil.html"
```

And this is our result:



Computername	DeviceID	SizeGB	FreeGB	UsedGB	PerFree
CHI-DB01	C:	39.90	18.46	21.44	46.27 %
CHI-DB01	D:	10.00	9.92	0.08	99.23 %
CHI-DB01	E:	10.00	9.88	0.12	98.80 %
CHI-DC01	C:	14.90	8.62	6.28	57.83 %
CHI-DC02	C:	11.90	4.92	6.98	41.38 %
CHI-EX01	C:	39.90	22.55	17.35	56.52 %
CHI-FP01	F:	100.00	99.91	0.09	99.91 %
CHI-FP01	C:	19.90	7.67	12.23	38.53 %

Good luck!